الغرض من هذا البرنامج هو **تقديم نظام متكامل لإدارة المبيعات والمخزون في متجر التجزئة**، حيث يمكن من خلاله:

1. **إضافة المنتجات** وتخزين معلوماتها الأساسية مثل الاسم، الرقم التعريفي، والسعر، وعدد الوحدات المتاحة في المخزون.
2. **إدارة عمليات البيع** للعملاء، مع استخدام **Queue** لمعالجة الطلبات بالترتيب الزمني لضمان خدمة العملاء بشكل منظم.
3. **تسجيل عمليات إعادة المنتجات أو الإرجاع** وتتبع آخر عمليات البيع باستخدام **Stack**، بحيث يمكن استرجاع آخر عملية بيع أو إرجاع بسرعة.
4. **توليد تقارير عن حالة المخزون** وعدد الوحدات المتاحة لكل منتج، وحالة المبيعات اليومية.
5. **حفظ واسترجاع البيانات من ملف خارجي** لضمان استمرار المعلومات بين جلسات التشغيل

C++

```cpp
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <queue>
#include <stack>
#include <string>
#include <sstream>
#include <iomanip>

using namespace std;

// File to store product data
const string DATA_FILE = "retail_products_cpp.txt";

// Struct for product summary
struct ProductSummary {
    string name;
    int availableStock;
    double price;
};
```

```
// Class for product
class Product {
public:
    string name;
    double price;
    int stock;
};

// Struct for sale/return transaction
struct SaleRequest {
    string productId;
    int quantity; // Positive for sale, negative for return
};

// Data structures
unordered_map<string, Product> products;
queue<SaleRequest> salesQueue; // FIFO for pending sales
stack<SaleRequest> transactionStack; // LIFO for last
transactions

// Function to add product
void addProduct() {
    string id, name;
    double price;
    int stock;

    cout << "Product ID: "; cin >> id;
    cin.ignore();
    cout << "Product Name: "; getline(cin, name);
    cout << "Price: "; cin >> price;
    cout << "Stock Quantity: "; cin >> stock;

    products[id] = Product{name, price, stock};
    cout << "Product added successfully.\n";
}

// Function to process sale
void processSale() {
```

```
    string id;
    int qty;
    cout << "Product ID to sell: "; cin >> id;
    cout << "Quantity: "; cin >> qty;

    if (products.find(id) == products.end()) {
        cout << "Product not found.\n";
        return;
    }

    if (products[id].stock < qty) {
        cout << "Not enough stock. Adding request to sales queue.\
n";
        salesQueue.push(SaleRequest{id, qty});
        return;
    }

    products[id].stock -= qty;
    transactionStack.push(SaleRequest{id, qty});
    cout << "Sold " << qty << " units of '" << products[id].name
<< "'.\n";
}

// Function to process return
void processReturn() {
    string id;
    int qty;
    cout << "Product ID to return: "; cin >> id;
    cout << "Quantity: "; cin >> qty;

    if (products.find(id) == products.end()) {
        cout << "Product not found.\n";
        return;
    }

    products[id].stock += qty;
    transactionStack.push(SaleRequest{id, -qty});
```

```
    cout << "Returned " << qty << " units of '" <<
products[id].name << "'.\n";

    // Process queued sales
    if (!salesQueue.empty()) {
        SaleRequest req = salesQueue.front();
        if (products[req.productId].stock >= req.quantity) {
            products[req.productId].stock -= req.quantity;
            salesQueue.pop();
            transactionStack.push(req);
            cout << "Queued sale processed: " << req.quantity << "
units of '"
                 << products[req.productId].name << "'.\n";
        }
    }
}

// Function to generate report
void generateReport() {
    cout << "\nProduct Inventory Report:\n";
    cout << left << setw(10) << "ID" << setw(20) << "Name"
         << setw(10) << "Stock" << setw(10) << "Price" << "\n";
    for (auto &p : products) {
        cout << setw(10) << p.first
             << setw(20) << p.second.name
             << setw(10) << p.second.stock
             << setw(10) << fixed << setprecision(2) <<
p.second.price
             << "\n";
    }
}

// Function to view last transaction
void viewLastTransaction() {
    if (transactionStack.empty()) {
        cout << "No transactions yet.\n";
        return;
    }
```

```
    SaleRequest t = transactionStack.top();
    string action = t.quantity > 0 ? "Sold" : "Returned";
    int qty = abs(t.quantity);
    cout << action << " " << qty << " units of '" <<
products[t.productId].name << "'.\n";
}

// Save products to file
void save() {
    ofstream file(DATA_FILE);
    for (auto &p : products) {
        file << p.first << "|" << p.second.name << "|" <<
p.second.price << "|" << p.second.stock << "\n";
    }
    file.close();
}

// Load products from file
void load() {
    ifstream file(DATA_FILE);
    if (!file.is_open()) return;

    string line;
    while (getline(file, line)) {
        stringstream ss(line);
        string id, name, priceStr, stockStr;
        getline(ss, id, '|');
        getline(ss, name, '|');
        getline(ss, priceStr, '|');
        getline(ss, stockStr, '|');

        Product p;
        p.name = name;
        p.price = stod(priceStr);
        p.stock = stoi(stockStr);
        products[id] = p;
    }
    file.close();
```

```
}

// Main menu loop
int main() {
    load();
    while (true) {
        cout << "\n1 Add Product  2 Sell Product  3 Return Product
4 Inventory Report  5 Last Transaction  6 Exit\n";
        int choice; cin >> choice;
        switch (choice) {
            case 1: addProduct(); break;
            case 2: processSale(); break;
            case 3: processReturn(); break;
            case 4: generateReport(); break;
            case 5: viewLastTransaction(); break;
            case 6: save(); return 0;
            default: cout << "Invalid option.\n"; break;
        }
    }
}

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_PRODUCTS 100
#define MAX_NAME_LEN 50
#define MAX_STACK 100
#define MAX_QUEUE 100
#define DATA_FILE "retail_products_c.txt"

// Structs
typedef struct {
    char name[MAX_NAME_LEN];
    double price;
    int stock;
} Product;

typedef struct {
```

```
    int productIndex;
    int quantity; // Positive for sale, negative for return
} Transaction;

// Global variables
Product products[MAX_PRODUCTS];
int productCount = 0;

Transaction transactionStack[MAX_STACK];
int stackTop = -1;

Transaction salesQueue[MAX_QUEUE];
int queueStart = 0;
int queueEnd = 0;

// Function prototypes
void addProduct();
void processSale();
void processReturn();
void generateReport();
void viewLastTransaction();
void saveData();
void loadData();

int main() {
    loadData();
    int choice;
    while (1) {
        printf("\n1 Add Product  2 Sell Product  3 Return Product  4 
Inventory Report  5 Last Transaction  6 Exit\n");
        scanf("%d", &choice);
        getchar(); // consume newline

        switch(choice) {
            case 1: addProduct(); break;
            case 2: processSale(); break;
            case 3: processReturn(); break;
            case 4: generateReport(); break;
```

```
            case 5: viewLastTransaction(); break;
            case 6: saveData(); exit(0);
            default: printf("Invalid option.\n");
        }
    }
    return 0;
}

void addProduct() {
    if (productCount >= MAX_PRODUCTS) {
        printf("Product limit reached.\n");
        return;
    }

    char name[MAX_NAME_LEN];
    double price;
    int stock;

    printf("Product Name: ");
    fgets(name, MAX_NAME_LEN, stdin);
    name[strcspn(name, "\n")] = 0; // Remove newline

    printf("Price: ");
    scanf("%lf", &price);
    printf("Stock Quantity: ");
    scanf("%d", &stock);
    getchar(); // consume newline

    strncpy(products[productCount].name, name,
MAX_NAME_LEN);
    products[productCount].price = price;
    products[productCount].stock = stock;

    productCount++;
    printf("Product added successfully.\n");
}

void processSale() {
    int index, qty;
```

```
    printf("Product index to sell (0-%d): ", productCount-1);
    scanf("%d", &index);
    if (index < 0 || index >= productCount) {
        printf("Invalid product index.\n");
        return;
    }

    printf("Quantity: ");
    scanf("%d", &qty);

    if (products[index].stock < qty) {
        printf("Not enough stock. Adding to sales queue.\n");
        if ((queueEnd+1) % MAX_QUEUE == queueStart) {
            printf("Queue is full.\n");
            return;
        }
        salesQueue[queueEnd].productIndex = index;
        salesQueue[queueEnd].quantity = qty;
        queueEnd = (queueEnd + 1) % MAX_QUEUE;
        return;
    }

    products[index].stock -= qty;
    stackTop++;
    transactionStack[stackTop].productIndex = index;
    transactionStack[stackTop].quantity = qty;

    printf("Sold %d units of '%s'.\n", qty, products[index].name);
}

void processReturn() {
    int index, qty;
    printf("Product index to return (0-%d): ", productCount-1);
    scanf("%d", &index);
    if (index < 0 || index >= productCount) {
        printf("Invalid product index.\n");
        return;
    }
```

```
    printf("Quantity: ");
    scanf("%d", &qty);

    products[index].stock += qty;
    stackTop++;
    transactionStack[stackTop].productIndex = index;
    transactionStack[stackTop].quantity = -qty;

    printf("Returned %d units of '%s'.\n", qty,
products[index].name);

    // Process queued sales
    if (queueStart != queueEnd) {
        Transaction req = salesQueue[queueStart];
        if (products[req.productIndex].stock >= req.quantity) {
            products[req.productIndex].stock -= req.quantity;
            stackTop++;
            transactionStack[stackTop] = req;
            queueStart = (queueStart + 1) % MAX_QUEUE;
            printf("Queued sale processed: %d units of '%s'.\n",
req.quantity, products[req.productIndex].name);
        }
    }
}

void generateReport() {
    printf("\nProduct Inventory Report:\n");
    printf("%-5s %-20s %-10s %-10s\n", "Index", "Name", "Stock",
"Price");
    for (int i = 0; i < productCount; i++) {
        printf("%-5d %-20s %-10d %-10.2f\n", i, products[i].name,
products[i].stock, products[i].price);
    }
}

void viewLastTransaction() {
    if (stackTop < 0) {
        printf("No transactions yet.\n");
        return;
```

```
    }
    Transaction t = transactionStack[stackTop];
    char* action = t.quantity > 0 ? "Sold" : "Returned";
    int qty = t.quantity > 0 ? t.quantity : -t.quantity;
    printf("%s %d units of '%s'\n", action, qty,
products[t.productIndex].name);
}

void saveData() {
    FILE *fp = fopen(DATA_FILE, "w");
    if (!fp) {
        printf("Error saving data.\n");
        return;
    }

    for (int i = 0; i < productCount; i++) {
        fprintf(fp, "%s|%lf|%d\n", products[i].name, products[i].price,
products[i].stock);
    }
    fclose(fp);
}

void loadData() {
    FILE *fp = fopen(DATA_FILE, "r");
    if (!fp) return;

    char line[256];
    while (fgets(line, sizeof(line), fp)) {
        char *token = strtok(line, "|");
        if (!token) continue;
        strncpy(products[productCount].name, token,
MAX_NAME_LEN);

        token = strtok(NULL, "|");
        products[productCount].price = atof(token);

        token = strtok(NULL, "|");
        products[productCount].stock = atoi(token);
```

```
        productCount++;
    }
    fclose(fp);
}
```

C#

```
using System;
using System.Collections.Generic;
using System.IO;

class RetailSystem
{
    const string DATA_FILE = "retail_products.txt";

    // Dictionary to store products (Product ID => Product)
    static Dictionary<string, Product> products = new
Dictionary<string, Product>();

    // Queue for sales requests (FIFO)
    static Queue<SaleRequest> salesQueue = new
Queue<SaleRequest>();

    // Stack for last transactions (LIFO)
    static Stack<SaleRequest> transactionStack = new
Stack<SaleRequest>();

    // Struct for product summary
    struct ProductSummary
    {
        public string Name;
        public int AvailableStock;
        public decimal Price;
    }

    class Product
    {
        public string Name;
        public decimal Price;
        public int Stock;
    }

    class SaleRequest
    {
```

```
        public string ProductId;
        public int Quantity;
    }

    static void AddProduct()
    {
        Console.Write("Product ID: ");
        string id = Console.ReadLine();
        Console.Write("Product Name: ");
        string name = Console.ReadLine();
        Console.Write("Price: ");
        decimal price = decimal.Parse(Console.ReadLine());
        Console.Write("Stock Quantity: ");
        int stock = int.Parse(Console.ReadLine());

        products[id] = new Product { Name = name, Price = price,
Stock = stock };
        Console.WriteLine("Product added successfully.");
    }

    static void ProcessSale()
    {
        Console.Write("Product ID to sell: ");
        string id = Console.ReadLine();
        Console.Write("Quantity: ");
        int quantity = int.Parse(Console.ReadLine());

        if (!products.ContainsKey(id))
        {
            Console.WriteLine("Product not found.");
            return;
        }

        if (products[id].Stock < quantity)
        {
            Console.WriteLine("Not enough stock. Adding request to
sales queue.");
```

```
            salesQueue.Enqueue(new SaleRequest { ProductId = id,
Quantity = quantity });
            return;
        }

        products[id].Stock -= quantity;
        var transaction = new SaleRequest { ProductId = id,
Quantity = quantity };
        transactionStack.Push(transaction);

        Console.WriteLine($"Sold {quantity} units of
'{products[id].Name}'.");
    }

    static void ProcessReturns()
    {
        Console.Write("Product ID to return: ");
        string id = Console.ReadLine();
        Console.Write("Quantity: ");
        int quantity = int.Parse(Console.ReadLine());

        if (!products.ContainsKey(id))
        {
            Console.WriteLine("Product not found.");
            return;
        }

        products[id].Stock += quantity;
        var transaction = new SaleRequest { ProductId = id,
Quantity = -quantity };
        transactionStack.Push(transaction);

        Console.WriteLine($"Returned {quantity} units of
'{products[id].Name}'.");

        // Process queued sales if stock available
        if (salesQueue.Count > 0)
        {
            var request = salesQueue.Peek();
```

```
            if (products[request.ProductId].Stock >= request.Quantity)
            {
                products[request.ProductId].Stock -= request.Quantity;
                salesQueue.Dequeue();
                transactionStack.Push(request);
                Console.WriteLine($"Queued sale processed: {request.Quantity} units of '{products[request.ProductId].Name}'.");
            }
        }
    }

    static void GenerateReport()
    {
        Console.WriteLine("\nProduct Inventory Report:");
        foreach (var p in products)
        {
            ProductSummary summary = new ProductSummary
            {
                Name = p.Value.Name,
                AvailableStock = p.Value.Stock,
                Price = p.Value.Price
            };
            Console.WriteLine($"Product ID: {p.Key}, Name: {summary.Name}, Stock: {summary.AvailableStock}, Price: {summary.Price:C}");
        }
    }

    static void ViewLastTransaction()
    {
        if (transactionStack.Count == 0)
        {
            Console.WriteLine("No transactions yet.");
            return;
        }

        var t = transactionStack.Peek();
```

```
        string action = t.Quantity > 0 ? "Sold" : "Returned";
        int qty = Math.Abs(t.Quantity);
        Console.WriteLine($"{action} {qty} units of
'{products[t.ProductId].Name}'");
    }

    static void Save()
    {
        using (StreamWriter sw = new StreamWriter(DATA_FILE))
        {
            foreach (var p in products)
            {
                sw.WriteLine($"{p.Key}|{p.Value.Name}|
{p.Value.Price}|{p.Value.Stock}");
            }
        }
    }
    static void Load()
    {
        if (!File.Exists(DATA_FILE)) return;

        foreach (var line in File.ReadAllLines(DATA_FILE))
        {
            var parts = line.Split('|');
            products[parts[0]] = new Product
            {
                Name = parts[1],
                Price = decimal.Parse(parts[2]),
                Stock = int.Parse(parts[3])
            };
        }
    }
    static void Main()
    {
        Load();
        while (true)
        {
            Console.WriteLine("\n1 Add Product  2 Sell Product  3
Return Product  4 Inventory Report  5 Last Transaction  6 Exit");
```

```
            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1": AddProduct(); break;
                case "2": ProcessSale(); break;
                case "3": ProcessReturns(); break;
                case "4": GenerateReport(); break;
                case "5": ViewLastTransaction(); break;
                case "6": Save(); return;
                default: Console.WriteLine("Invalid option."); break;
            }
        }
    }
}
```

VB.NET

```
Imports System.IO
Imports System.Collections.Generic

Module RetailSystem

    Const DATA_FILE As String = "retail_products_vb.txt"

    ' Dictionary to store products (Product ID -> Product)
    Dim products As New Dictionary(Of String, Product)

    ' Queue for pending sales (FIFO)
    Dim salesQueue As New Queue(Of SaleRequest)

    ' Stack for last transactions (LIFO)
    Dim transactionStack As New Stack(Of SaleRequest)

    ' Structure for product summary
    Structure ProductSummary
        Public Name As String
        Public AvailableStock As Integer
        Public Price As Decimal
```

```
    End Structure

    Class Product
        Public Name As String
        Public Price As Decimal
        Public Stock As Integer
    End Class

    Class SaleRequest
        Public ProductId As String
        Public Quantity As Integer ' Positive for sale, negative for
return
    End Class

    Sub AddProduct()
        Console.Write("Product ID: ")
        Dim id As String = Console.ReadLine()
        Console.Write("Product Name: ")
        Dim name As String = Console.ReadLine()
        Console.Write("Price: ")
        Dim price As Decimal = Decimal.Parse(Console.ReadLine())
        Console.Write("Stock Quantity: ")
        Dim stock As Integer = Integer.Parse(Console.ReadLine())

        products(id) = New Product With {.Name = name, .Price =
price, .Stock = stock}
        Console.WriteLine("Product added successfully.")
    End Sub

    Sub ProcessSale()
        Console.Write("Product ID to sell: ")
        Dim id As String = Console.ReadLine()
        Console.Write("Quantity: ")
        Dim qty As Integer = Integer.Parse(Console.ReadLine())

        If Not products.ContainsKey(id) Then
            Console.WriteLine("Product not found.")
            Return
```

```
        End If

        If products(id).Stock < qty Then
            Console.WriteLine("Not enough stock. Adding to sales
queue.")
            salesQueue.Enqueue(New SaleRequest With {.ProductId
= id, .Quantity = qty})
            Return
        End If

        products(id).Stock -= qty
        transactionStack.Push(New SaleRequest With {.ProductId =
id, .Quantity = qty})
        Console.WriteLine($"Sold {qty} units of
'{products(id).Name}'.")
    End Sub

    Sub ProcessReturn()
        Console.Write("Product ID to return: ")
        Dim id As String = Console.ReadLine()
        Console.Write("Quantity: ")
        Dim qty As Integer = Integer.Parse(Console.ReadLine())

        If Not products.ContainsKey(id) Then
            Console.WriteLine("Product not found.")
            Return
        End If

        products(id).Stock += qty
        transactionStack.Push(New SaleRequest With {.ProductId =
id, .Quantity = -qty})
        Console.WriteLine($"Returned {qty} units of
'{products(id).Name}'.")

        ' Process queued sales
        If salesQueue.Count > 0 Then
            Dim req As SaleRequest = salesQueue.Peek()
            If products(req.ProductId).Stock >= req.Quantity Then
```

```
                products(req.ProductId).Stock -= req.Quantity
                transactionStack.Push(req)
                salesQueue.Dequeue()
                Console.WriteLine($"Queued sale processed:
{req.Quantity} units of '{products(req.ProductId).Name}'.")
            End If
        End If
    End Sub

    Sub GenerateReport()
        Console.WriteLine(vbCrLf & "Product Inventory Report:")
        For Each kvp In products
            Dim summary As New ProductSummary With {
                .Name = kvp.Value.Name,
                .AvailableStock = kvp.Value.Stock,
                .Price = kvp.Value.Price
            }
            Console.WriteLine($"Product ID: {kvp.Key}, Name:
{summary.Name}, Stock: {summary.AvailableStock}, Price:
{summary.Price:C}")
        Next
    End Sub

    Sub ViewLastTransaction()
        If transactionStack.Count = 0 Then
            Console.WriteLine("No transactions yet.")
            Return
        End If

        Dim t As SaleRequest = transactionStack.Peek()
        Dim action As String = If(t.Quantity > 0, "Sold", "Returned")
        Dim qty As Integer = Math.Abs(t.Quantity)
        Console.WriteLine($"{action} {qty} units of
'{products(t.ProductId).Name}'")
    End Sub

    Sub SaveData()
        Using sw As New StreamWriter(DATA_FILE)
```

```
            For Each kvp In products
                sw.WriteLine($"{kvp.Key}|{kvp.Value.Name}|
{kvp.Value.Price}|{kvp.Value.Stock}")
            Next
        End Using
    End Sub

    Sub LoadData()
        If Not File.Exists(DATA_FILE) Then Return
        For Each line As String In File.ReadAllLines(DATA_FILE)
            Dim parts() As String = line.Split("|"c)
            products(parts(0)) = New Product With {
                .Name = parts(1),
                .Price = Decimal.Parse(parts(2)),
                .Stock = Integer.Parse(parts(3))
            }
        Next
    End Sub

    Sub Main()
        LoadData()
        While True
            Console.WriteLine(vbCrLf & "1 Add Product  2 Sell Product
3 Return Product  4 Inventory Report  5 Last Transaction  6 Exit")
            Dim choice As String = Console.ReadLine()
            Select Case choice
                Case "1" : AddProduct()
                Case "2" : ProcessSale()
                Case "3" : ProcessReturn()
                Case "4" : GenerateReport()
                Case "5" : ViewLastTransaction()
                Case "6" : SaveData() : Exit While
                Case Else : Console.WriteLine("Invalid option.")
            End Select
        End While
    End Sub

End Module
```

Visual Basic (VB6 / Classic VB)

```
Option Explicit

' Constants
Const MAX_PRODUCTS As Integer = 100
Const MAX_STACK As Integer = 100
Const MAX_QUEUE As Integer = 100
Const DATA_FILE As String = "retail_products_vb.txt"

' Types
Type Product
    Name As String
    Price As Double
    Stock As Integer
End Type

Type SaleRequest
    ProductIndex As Integer
    Quantity As Integer ' Positive for sale, negative for return
End Type

' Global Variables
Dim products(MAX_PRODUCTS) As Product
Dim productCount As Integer

Dim transactionStack(MAX_STACK) As SaleRequest
Dim stackTop As Integer

Dim salesQueue(MAX_QUEUE) As SaleRequest
Dim queueStart As Integer
Dim queueEnd As Integer

' --- Procedures ---

Sub AddProduct()
    If productCount >= MAX_PRODUCTS Then
        MsgBox "Product limit reached."
        Exit Sub
```

```
    End If

    Dim name As String
    Dim price As Double
    Dim stock As Integer

    name = InputBox("Enter Product Name:")
    price = CDbl(InputBox("Enter Price:"))
    stock = CInt(InputBox("Enter Stock Quantity:"))

    products(productCount).Name = name
    products(productCount).Price = price
    products(productCount).Stock = stock

    productCount = productCount + 1
    MsgBox "Product added successfully."
End Sub

Sub ProcessSale()
    Dim index As Integer
    Dim qty As Integer
    index = CInt(InputBox("Enter Product Index (0-" &
(productCount - 1) & "):"))
    If index < 0 Or index >= productCount Then
        MsgBox "Invalid product index."
        Exit Sub
    End If

    qty = CInt(InputBox("Enter Quantity:"))

    If products(index).Stock < qty Then
        MsgBox "Not enough stock. Adding to sales queue."
        salesQueue(queueEnd).ProductIndex = index
        salesQueue(queueEnd).Quantity = qty
        queueEnd = (queueEnd + 1) Mod MAX_QUEUE
        Exit Sub
    End If
```

```
    products(index).Stock = products(index).Stock - qty
    stackTop = stackTop + 1
    transactionStack(stackTop).ProductIndex = index
    transactionStack(stackTop).Quantity = qty

    MsgBox "Sold " & qty & " units of '" & products(index).Name &
"'."
End Sub

Sub ProcessReturn()
    Dim index As Integer
    Dim qty As Integer
    index = CInt(InputBox("Enter Product Index (0-" &
(productCount - 1) & "):"))
    If index < 0 Or index >= productCount Then
        MsgBox "Invalid product index."
        Exit Sub
    End If

    qty = CInt(InputBox("Enter Quantity:"))
    products(index).Stock = products(index).Stock + qty

    stackTop = stackTop + 1
    transactionStack(stackTop).ProductIndex = index
    transactionStack(stackTop).Quantity = -qty

    MsgBox "Returned " & qty & " units of '" &
products(index).Name & "'."

    ' Process queued sales
    If queueStart <> queueEnd Then
        Dim req As SaleRequest
        req = salesQueue(queueStart)
        If products(req.ProductIndex).Stock >= req.Quantity Then
            products(req.ProductIndex).Stock =
products(req.ProductIndex).Stock - req.Quantity
            stackTop = stackTop + 1
            transactionStack(stackTop) = req
```

```
            queueStart = (queueStart + 1) Mod MAX_QUEUE
            MsgBox "Queued sale processed: " & req.Quantity & "
units of '" & products(req.ProductIndex).Name & "'."
        End If
    End If
End Sub

Sub GenerateReport()
    Dim i As Integer
    Dim report As String
    report = "Product Inventory Report:" & vbCrLf
    For i = 0 To productCount - 1
        report = report & "Index: " & i & ", Name: " &
products(i).Name & _
                 ", Stock: " & products(i).Stock & ", Price: " &
Format(products(i).Price, "Currency") & vbCrLf
    Next
    MsgBox report
End Sub

Sub ViewLastTransaction()
    If stackTop < 0 Then
        MsgBox "No transactions yet."
        Exit Sub
    End If
    Dim t As SaleRequest
    t = transactionStack(stackTop)
    Dim action As String
    action = IIf(t.Quantity > 0, "Sold", "Returned")
    Dim qty As Integer
    qty = Abs(t.Quantity)
    MsgBox action & " " & qty & " units of '" &
products(t.ProductIndex).Name & "'."
End Sub

Sub SaveData()
    Dim fnum As Integer
    fnum = FreeFile
```

```
    Open DATA_FILE For Output As #fnum
    Dim i As Integer
    For i = 0 To productCount - 1
        Print #fnum, products(i).Name & "|" & products(i).Price & "|"
& products(i).Stock
    Next i
    Close #fnum
End Sub

Sub LoadData()
    Dim fnum As Integer
    If Dir(DATA_FILE) = "" Then Exit Sub
    fnum = FreeFile
    Open DATA_FILE For Input As #fnum
    Dim line As String
    Do While Not EOF(fnum)
        Line Input #fnum, line
        Dim parts() As String
        parts = Split(line, "|")
        products(productCount).Name = parts(0)
        products(productCount).Price = CDbl(parts(1))
        products(productCount).Stock = CInt(parts(2))
        productCount = productCount + 1
    Loop
    Close #fnum
End Sub

Sub Main()
    stackTop = -1
    queueStart = 0
    queueEnd = 0

    LoadData()

    Dim choice As Integer
    Do
        choice = CInt(InputBox("1 Add Product  2 Sell Product  3
Return Product  4 Inventory Report  5 Last Transaction  6 Exit"))
```

```
        Select Case choice
            Case 1: AddProduct
            Case 2: ProcessSale
            Case 3: ProcessReturn
            Case 4: GenerateReport
            Case 5: ViewLastTransaction
            Case 6: SaveData: Exit Do
            Case Else: MsgBox "Invalid option."
        End Select
    Loop
End Sub

' Run Main
Call Main
```